— GUIDE —

Assignment 3: Solve Nonograms

AI-1 Systems Project (Summer Semester 2024)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

This document is intended to help you solve the assignment "Assignment 3: Solve Nonograms" [AS]. You do not have to read it, but we do recommend to at least take a look at the tips and common issues.

1 A few tips

1. As a starting point, you can focus on classical nonograms (rectangular grid and just one non-background color). That should also get you many points already. When you have a good understanding of how to solve them, moving on to the other nonograms should be more manageable.
2. This assignment benefits from discussion. You should ask for help, e.g. at the office hours or in the assignment room on Matrix – especially if you are stuck at the early stages (which happens to many students).
3. Converting formulae to CNF efficiently takes a bit of practice. We have included some tips and exercises in Section 4 to help you get started.
4. For debugging purposes, you might want to make your own, small nonograms (e.g. 2×2 or 2×3 rectangles). If the small nonograms work, but you still have problems with the larger ones, debugging gets trickier. Here are two strategies for re-producing bugs on a smaller problem if the nonogram is rectangular:
   (a) *Case 1:* You do not have enough constraints, i.e. you find a solution that should be impossible according to the clues. In this case, you can reduce the problem to the problematic row/column. Let us assume that the issue is in a row. Then you can make a new nonogram that only consists of that row, using the same row clue, but enforcing the wrong solution with custom column hints. This should be unsatisfiable, but (because of the bug), you should get a solution anyway. You can

then carefully examine the solution and find out what constraint was missing/did not work.

(b) *Case 2:* You have too many constraints, i.e. you do not find any solution, even though the problem should be solvable. This strategy requires you to know the correct solution. You can test each row/column separately. To test a row, you provide the row clue and use the column clues to enforce the correct solution. This should be satisfiable, but for at least one row, or column, it should be unsatisfiable because of the bug. Once you have identified the problematic row/column, you can try to examine what constraint prevents the correct solution.

2 Common issues and misconceptions

1. For some approaches, it is very important that the conversion to CNF is efficient. Section 4 contains some tips and exercises to help you get started. If you use a library, it may use a less efficient conversion, which can lead to severe performance issues.
2. People usually expect that the SAT solver will be the bottleneck. But for very inefficient encodings, it is more common that the encoder becomes the bottleneck. Additionally, it appears that SAT solvers tend to process more compact encodings faster.
3. Section 3 sketches four very different approaches. A common misconception is that approaches 2–4 are just different ways to generate the possible combinations for approach 1. That is not true: approaches 2–4 do not require to explicitly list all possible combinations.

3 Encoding nonograms as SAT problems

In this section, we will sketch different ways how a traditional nonogram can be encoded has a SAT problem. Some are easier to implement than others and some perform much better than others (at least for large nonograms). For didactic reasons, we will leave the analysis for you (see the assignment sheet) and only sketch the approaches.

As a running example, we will explore the encoding of a single clue, `2a 1a 2a` for a 10-cell row in a traditional (i.e. single-colored) nonogram. We will use the variables $C_1, \dots, C_{10}$ to indicate for each cell if it should be colored. When we get a variable assignment from the SAT solver, we can simply check what values are assigned to $C_1, \dots, C_{10}$ to know how we should color the cells. The challenge is now to find a CNF formula that only allows variable

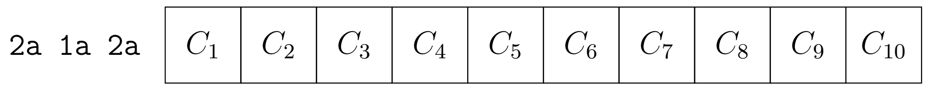

2a 1a 2a
C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10}

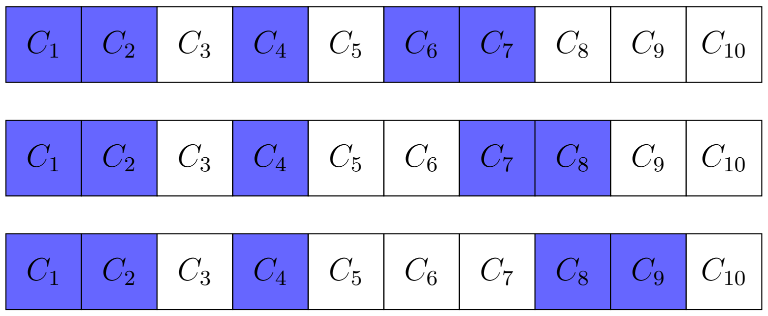
Figure 1: Running example for demonstrating the encodings. C_i indicates if the i-th cell should be colored.

assignments that are compatible with the clue. The following subsections will sketch different approaches to achieve this.

For a complete nonogram, we would have to encode all clues in the same way. (but of course we have to be careful that we have a different color variable for each cell).

3.1 Approach 1: Listing possible arrangements

This is the most obvious approach and you will find similar solutions when searching for "SAT-based nonogram solver" in the internet. In this approach, you simply list all the possible arrangements and state that one of them must be true. The possible arrangements for the example clue would be:


C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10}

etc.

We can easily express this as a DNF formula:

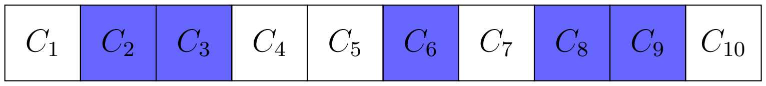
$$\begin{aligned} &(C_1 \wedge C_2 \wedge \neg C_3 \wedge C_4 \wedge \neg C_5 \wedge C_6 \wedge C_7 \wedge \neg C_8 \wedge \neg C_9 \wedge \neg C_{10}) \\ \vee\ &(C_1 \wedge C_2 \wedge \neg C_3 \wedge C_4 \wedge \neg C_5 \wedge \neg C_6 \wedge C_7 \wedge C_8 \wedge \neg C_9 \wedge \neg C_{10}) \\ \vee\ &(C_1 \wedge C_2 \wedge \neg C_3 \wedge C_4 \wedge \neg C_5 \wedge \neg C_6 \wedge \neg C_7 \wedge C_8 \wedge C_9 \wedge \neg C_{10}) \\ \vee\ &\ldots \end{aligned}$$

To convert the formula to CNF, it helps to introduce one helper variable for each arrangement (see also Section 4.2).

3.2 Approach 2: Use variables to denote each block start

In the example clue, we have three blocks, which we will refer to as α, β and γ. We will introduce variables S_ξ^i for each $\xi \in \{\alpha, \beta, \gamma\}$ and $i \in \{1, \dots, 10\}$ to indicate the ξ block starts at cell i.

For example, in the arrangement



the α block would start in cell 2, the β block in cell 6 and the γ block in cell 8. We would therefore represent the arrangement as $S_\alpha^2 \wedge S_\beta^6 \wedge S_\gamma^8$ (and all other S_ξ^i would be false).

Now we have to do two things:

1. link the block starts to cell colors and
2. make sure that we only get valid combinations of block starts.

Linking block starts to cell colors We can link the block starts to cell colors with simple implications. For example, if the α block starts at position 1, then cells 1 and 2 must be colored (because, according to the clue, it is a block of length 2), i.e. $S_\alpha^1 \Rightarrow (C_1 \wedge C_2)$. Similarly, $S_\alpha^2 \Rightarrow (C_2 \wedge C_3)$ etc. While such constraints make sure that all cells that belong to a block are colored, we also need additional constraints to make sure that cells are *only* colored if they belong to a block. For example, if cell 7 is colored, then it must be be part of the the α block, the β block or the γ block[1], which means that either the α block starts at 6 or 7, or the β block starts at 7 or the γ block starts at 7:

$$(C_7 \Rightarrow (S_\alpha^6 \vee S_\alpha^7 \vee S_\beta^7 \vee S_\gamma^7))$$

Valid combinations of block starts Making sure we only get valid combinations of block starts is a bit trickier. First of all, we can use the exclusive or ($\oplus$) to make sure that each block has *exactly one* start:

$$S_\alpha^1 \oplus S_\alpha^2 \oplus S_\alpha^3 \oplus \dots$$

We can do the same for β and γ. The last step is now to make sure that the blocks are appropriately spaced. There are different ways to do this, and we will sketch three variants. You only need to implement one of them.

[1]Technically, it cannot be part of the α block. It is up to you whether you want to include "impossible" block starts in your encoding or not.

Variant 1 – do not use Similarly to approach 1, we simply list all possible arrangements:

$$(S_\alpha^1 \wedge S_\beta^4 \wedge S_\gamma^6) \vee (S_\alpha^1 \wedge S_\beta^4 \wedge S_\gamma^7) \vee (S_\alpha^1 \wedge S_\beta^4 \wedge S_\gamma^8) \vee (S_\alpha^1 \wedge S_\beta^4 \wedge S_\gamma^9) \vee (S_\alpha^1 \wedge S_\beta^5 \wedge S_\gamma^7) \vee \dots$$

Do not use this variant because it is basically the same as approach 1.

Variant 2 We make sure that if one block starts at a particular position, the next block will start sufficiently late. For example, if the α block starts at position 1, then the β block should start at position 4 or 5 or ...:

$$\begin{aligned} &(S_\alpha^1 \Rightarrow (S_\beta^4 \vee S_\beta^5 \vee \dots)) \\ \wedge\ &(S_\alpha^2 \Rightarrow (S_\beta^5 \vee S_\beta^6 \vee \dots)) \\ \wedge\ &\dots \\ \wedge\ &(S_\beta^1 \Rightarrow (S_\gamma^3 \vee S_\gamma^4 \vee \dots)) \\ \wedge\ &(S_\beta^2 \Rightarrow (S_\gamma^4 \vee S_\gamma^5 \vee \dots)) \\ \wedge\ &\dots \end{aligned}$$

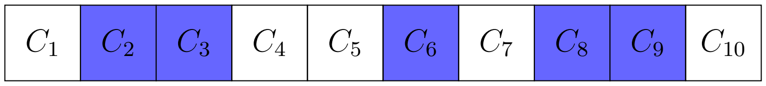
Note: It's a design choice whether you want to create impossible block starts like S_β^1 (the SAT solver would probably discard them very quickly).

Variant 3 We make sure that if one block starts at a particular position, the next block will not start too early. For example, if the α block starts at position 1, then the β block should not start at position 1 or 2 or 3:

$$\begin{aligned} &(S_\alpha^1 \Rightarrow (\neg S_\beta^1 \wedge \neg S_\beta^2 \wedge \neg S_\beta^3)) \\ \wedge\ &(S_\alpha^2 \Rightarrow (\neg S_\beta^1 \wedge \neg S_\beta^2 \wedge \neg S_\beta^3 \wedge \neg S_\beta^4)) \\ \wedge\ &\dots \\ \wedge\ &(S_\beta^1 \Rightarrow (\neg S_\gamma^1 \wedge \neg S_\gamma^2)) \\ \wedge\ &(S_\beta^2 \Rightarrow (\neg S_\gamma^1 \wedge \neg S_\gamma^2 \wedge \neg S_\gamma^3)) \\ \wedge\ &\dots \end{aligned}$$

3.3 Approach 3: Enclose blocks

Like in approach 2, we will name the three blocks in the clue α, β and γ. For each block $\xi \in \{\alpha, \beta, \gamma\}$ and each cell $i \in \{1, \ldots, 10\}$ we will now introduce two variables: A^i_ξ and B^i_ξ. A^i_ξ indicates that the block ξ *starts after* cell i and B^i_ξ indicates that the block ξ *ends before* cell i. For example, the arrangement



would be encoded as

$$\begin{aligned}
&A^1_\alpha \wedge \neg A^2_\alpha \wedge \neg A^3_\alpha \wedge \neg A^4_\alpha \wedge \neg A^5_\alpha \wedge \neg A^6_\alpha \wedge \neg A^7_\alpha \wedge \neg A^8_\alpha \wedge \neg A^9_\alpha \wedge \neg A^{10}_\alpha \\
\wedge\ &\neg B^1_\alpha \wedge \neg B^2_\alpha \wedge \neg B^3_\alpha \wedge B^4_\alpha \wedge B^5_\alpha \wedge B^6_\alpha \wedge B^7_\alpha \wedge B^8_\alpha \wedge B^9_\alpha \wedge B^{10}_\alpha \\
\wedge\ &A^1_\beta \wedge A^2_\beta \wedge A^3_\beta \neg A^4_\beta \wedge A^5_\beta \wedge \neg A^6_\beta \wedge \neg A^7_\beta \wedge \neg A^8_\beta \wedge \neg A^9_\beta \wedge \neg A^{10}_\beta \\
\wedge\ &\ldots
\end{aligned}$$

Now, cell i should be colored if and only if it belongs to one of the blocks, i.e. if one of the blocks does not start after i and does not end before i. So, in our example, we would e.g. have for cell 5

$$C_5 \Leftrightarrow ((\neg A^5_\alpha \wedge \neg B^5_\alpha) \vee (\neg A^5_\beta \wedge \neg B^5_\beta) \vee (\neg A^5_\gamma \wedge \neg B^5_\gamma))$$

We also put some basic constraints on our A and B variables. If a block starts after cell i, it also starts after cell $i-1$, i.e.

$$(A^2_\alpha \Rightarrow A^1_\alpha) \wedge (A^3_\alpha \Rightarrow A^2_\alpha) \wedge (A^4_\alpha \Rightarrow A^3_\alpha) \wedge \ldots \wedge (A^2_\beta \Rightarrow A^1_\beta) \wedge (A^3_\beta \Rightarrow A^2_\beta) \wedge (A^4_\beta \Rightarrow A^3_\beta) \wedge \ldots$$

We can treat the B variables analogously. As "$\Rightarrow$" is transitive, the above formula also implies e.g. $A^4_\alpha \Rightarrow A^2_\alpha$.

At last, we also have to make sure that the blocks have the right length. For example, the α block should have length 2 according to the clue. This means that if the block does not start after i (i.e. it starts at i or before i), then it should end before $i+2$. Concretely, that gives us

$$((\neg A^1_\alpha) \Rightarrow B^3_\alpha) \wedge ((\neg A^2_\alpha) \Rightarrow B^4_\alpha) \wedge ((\neg A^3_\alpha) \Rightarrow B^5_\alpha) \wedge \ldots$$

3.4 Approach 4: Make an automaton

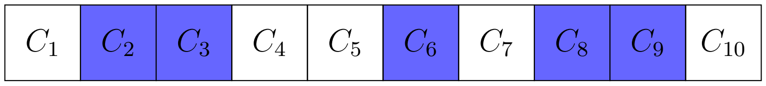
In this approach, we will effectively create something like a finite automaton[2]. The input is a sequence of cell colors and the automaton will only accept sequences that match the clue. Each state intuitively describes the role of the previously entered cell. For the example clue, we would have the following states (again calling the 3 blocks α, β and γ):

State	**Role of previously entered cell**
S	Cell before the first (α) block
α^1	First cell of the α block
α^2	Second cell of the α block
$G^{\alpha\beta}$	Cell in the gap between the α and the β block.
β^1	First cell of the β block
$G^{\beta\gamma}$	Cell in the gap between the β and the γ block.
γ^1	First cell of the γ block
γ^2	Second cell of the γ block
E	Cell after the γ block

Writing c for a colored cell and $\neg c$ for an uncolored cell, the full automaton is:

$$\begin{tikzcd}
{} \arrow[r] & S \arrow[r, "c"] \arrow[loop below, "\neg c"] & \alpha^1 \arrow[r, "c"] & \alpha^2 \arrow[r, "\neg c"] & G^{\alpha\beta} \arrow[r, "c"] \arrow[loop below, "\neg c"] & \beta^1 \arrow[r, "\neg c"] & G^{\beta\gamma} \arrow[r, "c"] \arrow[loop below, "\neg c"] & \gamma^1 \arrow[r, "c"] & \gamma^2 \arrow[r, "\neg c"] & E \arrow[loop below, "\neg c"]
\end{tikzcd}$$

The automaton starts in state S and accepts the input if it ends up in state γ^2 or E. It would, for example, accept the sequence $\neg c, c, c, \neg c, \neg c, c, \neg c, c, c, \neg c$, which corresponds to the arrangement



The state sequence for that input is $S, \alpha^1, \alpha^2, G^{\alpha\beta}, G^{\alpha\beta}, \beta^1, G^{\beta\gamma}, \gamma^1, \gamma^2, E$. An input starting with c, c, c (which is not allowed according to the clue) would get rejected because a c input is not allowed in state α^2.

To represent the automaton in propositional logic, we will introduce variables Σ_i indicating that we are in state Σ after processing the input until (and including) cell i. For

[2]For simplicity, we do not exactly follow the traditional definition of a deterministic finite automaton. In particular, the transition function is not total.

example, if the first cell is colored (C_1 is true), then we will be in state α^1 after processing it. If it is not colored, on the other hand, we will remain in state S. That gives us the following formulae:

$$\begin{aligned} C_1 &\Rightarrow \alpha_1^1 \\ \neg C_1 &\Rightarrow S_1 \end{aligned}$$

If we are now in state S (i.e. S_1 is true), then, depending on the next cell color, we will either end up in state S or α^1:

$$\begin{aligned} (S_1 \wedge \neg C_2) &\Rightarrow S_2 \\ (S_1 \wedge C_2) &\Rightarrow \alpha_2^1 \end{aligned}$$

But if we were in state α^1 (i.e. α_1^1 is true), then the next cell has to be colored (C_2 must be true) and we will end up in state α^2:

$$\alpha_1^1 \Rightarrow C_2 \wedge \alpha_2^2$$

Here are the formulae for cell 3:

$$\begin{aligned} (S_2 \wedge \neg C_3) &\Rightarrow S_3 \\ (S_2 \wedge C_3) &\Rightarrow \alpha_3^1 \\ \alpha_2^1 &\Rightarrow (C_3 \wedge \alpha_3^2) \\ \alpha_2^2 &\Rightarrow (\neg C_3 \wedge G_3^{\alpha\beta}) \end{aligned}$$

We can do the same for the remaining cells. However, we must ensure that, after processing the last cell (cell 10), we are in state γ^2 or E, so we should add one more constraint:

$$\gamma_{10}^2 \vee E_{10}$$

Note: If you also support impossible states in your encoding (e.g. γ_1^1), you might have to add additional constraints to make sure that the automaton will only be in one state at each cell. Otherwise, your automaton might end up in two states simultaneously for a cell (one valid and one impossible), and then continue with the impossible state to reach an impossible terminal state. A better approach would be to exclude any impossible states in the first place.

4 Translating to CNF: Tips and exercises

SAT solvers expect formulae in CNF. When you try to encode the nonogram as a SAT problem, you might come up with formulae that are not in CNF. To help you get started, we thought it might be useful if we list some formulae for you to convert to CNF as an exercise. Your conversions should be efficient (otherwise, your nonogram encoding might not scale).

4.1 Exercises

Here are some formulae for you to convert to CNF in increasing difficulty:

1. $A \Rightarrow B$
2. $A \Rightarrow (B \vee C)$
3. $A \Leftrightarrow B$
4. $(\neg A) \Rightarrow (B \vee \neg C \vee D)$
5. $A \Rightarrow (B \wedge C)$
6. $A \Rightarrow (B \wedge C \wedge D \wedge E)$
7. $(A \wedge B) \Rightarrow C$
8. $(A \wedge B \wedge C \wedge D) \Rightarrow E$
9. $(A \wedge B \wedge C \wedge D) \Rightarrow (E \vee F \vee G)$

For the following formulae, it helps to introduce helper variables (see also next section):

1. $(A \vee B \vee C \vee D) \Rightarrow (E \wedge F \wedge G \wedge H)$
2. $(A \wedge B \wedge C) \vee (D \wedge E \wedge F) \vee (G \wedge H \wedge I)$

4.2 Using helper variables

Sometimes, it is helpful to introduce further **helper variables** to make the translation more efficient. For example, the formula $\varphi := (X \wedge Y) \vee Z$ could be translated to the CNF formula $\psi := (A \vee Z) \wedge (\neg A \vee X) \wedge (\neg A \vee Y)$. While φ and ψ are technically not equivalent, they are equisatisfiable. More concretely: for any assignment that satisfies φ, we can find an assignment that also satisfies ψ. Furthermore, any assignment that satisfies ψ also satisfies φ.

To show you a more relevant example, let us consider the formula

$$(X_1 \vee X_2 \vee X_3 \vee X_4) \Rightarrow (Y_1 \wedge Y_2 \wedge Y_3 \wedge Y_4)$$

We can see that whenever X_1 is true, Y_1 must also be true ($X_1 \Rightarrow Y_1$). Similarly, $X_1 \Rightarrow Y_2$, $X_1 \Rightarrow Y_3$ and $X_1 \Rightarrow Y_4$. Furthermore, $X_2 \Rightarrow Y_1$ and so on. Since $A \Rightarrow B$ is the same as $\neg A \vee B$, we can translate the formula to the following CNF formula[3]:

$$\begin{aligned}
&(\neg X_1 \vee Y_1) \wedge (\neg X_1 \vee Y_2) \wedge (\neg X_1 \vee Y_3) \wedge (\neg X_1 \vee Y_4) \wedge \\
&(\neg X_2 \vee Y_1) \wedge (\neg X_2 \vee Y_2) \wedge (\neg X_2 \vee Y_3) \wedge (\neg X_2 \vee Y_4) \wedge \\
&(\neg X_3 \vee Y_1) \wedge (\neg X_3 \vee Y_2) \wedge (\neg X_3 \vee Y_3) \wedge (\neg X_3 \vee Y_4) \wedge \\
&(\neg X_4 \vee Y_1) \wedge (\neg X_4 \vee Y_2) \wedge (\neg X_4 \vee Y_3) \wedge (\neg X_4 \vee Y_4)
\end{aligned}$$

This is not particularly efficient: If we generalize the original formula to $(X_1 \vee \ldots \vee X_n) \Rightarrow (Y_1 \wedge \ldots \wedge Y_n)$, we would get n^2 clauses. We can reduce this to $2n + 1$ if we introduce two helper variables, L and R, where L is intuitively the antecedent of the implication (the left hand side) and R is the consequent (the right hand side). That lets us re-write the formula as $L \Rightarrow R$. Now, if X_1 is true, then the antecedent must be true, i.e. $X_1 \Rightarrow L$. Similarly, $X_2 \Rightarrow L$ and so on. And if the consequent, i.e. R, is true, then Y_1 must be true ($R \Rightarrow Y_1$). Similarly, $R \Rightarrow Y_2$ and so on. So we get the equisatisfiable formula[4]

$$\begin{aligned}
&(\neg L \vee R) \wedge \\
&(\neg X_1 \vee L) \wedge (\neg X_2 \vee L) \wedge (\neg X_3 \vee L) \wedge (\neg X_4 \vee L) \wedge \\
&(\neg R \vee Y_1) \wedge (\neg R \vee Y_2) \wedge (\neg R \vee Y_3) \wedge (\neg R \vee Y_4)
\end{aligned}$$

This approach is similar to the Tseytin transformation (see e.g. [TT]).

[3]Actually, I only argued that the following formula must be true if the original formula is true. For equivalence, we need the other direction as well. If you stare at it for a while, I hope that you would agree that the other direction works as well.

[4]Like with the previous translation, we should make sure that the other direction works as well.